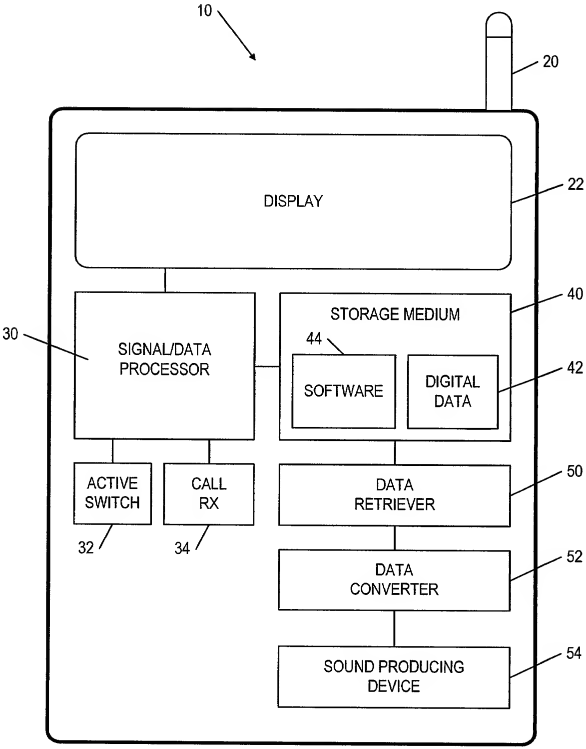
10
20
DISPLAY
22
SIGNAL/DATA PROCESSOR
30
STORAGE MEDIUM
40
44
SOFTWARE
DIGITAL DATA
42
ACTIVE SWITCH
CALL RX
32
34
DATA RETRIEVER
50
DATA CONVERTER
52
SOUND PRODUCING DEVICE
54

FIG. 1

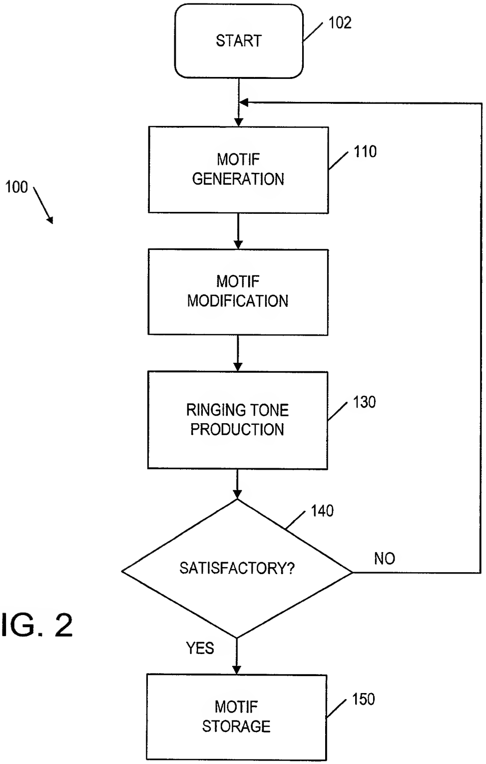
START
102
100
MOTIF GENERATION
110
MOTIF MODIFICATION
RINGING TONE PRODUCTION
130
140
SATISFACTORY?
NO
YES
MOTIF STORAGE
150

FIG. 2

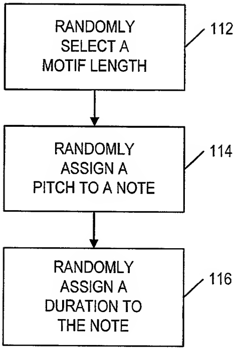
RANDOMLY SELECT A MOTIF LENGTH
112
RANDOMLY ASSIGN A PITCH TO A NOTE
114
RANDOMLY ASSIGN A DURATION TO THE NOTE
116

FIG. 3

```
from mid import *
import random

## The tst(leng,filename) function is the "main",
## calling it executes the program and generates
## musical data
## Program is based on the idea of repeated
## motive, which may include simple motives and
## and which can be transformed pitchwise and
## rhythmically

## ABOUT DATA STRUCTURES:
## Two simple array-like lists are used to store
## the starting points (from previous notes) and
## and the pitches of the notes. It is necessary
## to be able to insert and delete from these
## lists
## These lists are initialized in the tst-
## function and after that modified using several
## processing functions

def tst(leng,filename):
##Initialization of the motives and several
## control parameters like initial motive length
    mym = midilist()
    mym.filename = filename
    motive = []
    transform = []
    rhythmo = []

## Motive transformation data-list
## initialization, data is stored pairvise
## first representing the transformation
## maximum and second in pair the transformation
## amount
```

FIG. 4a

```
        transfolen = random.randint(1,3)
        transfotem,transfomodtem = 0,0
        for n in range(transfolen):
            transfotem = random.randint(-2,3)
            transfomodtem = random.randint(-4,4)
            transform.insert(0,transfotem)
            transform.insert(0,transfomodtem)

## More control parameters, motive lenght, whole
## piece transposition and non-legatones
        motilen = random.randint(2,7)
        transpo = random.randint(-12,12)
        transpo = 0
        randura = random.randint(1,5)

        motitem = 4
        startp = 0
        duratio = 12

## Actual motive and rhythm initialization
        for n in range(motilen):
            motitem = random.randint(0,24)
            motive.insert(0,motitem)

## Rhythmic motive (rhythmo) initialization, it
## is assumed that 96 is something like
## 100 - 200 milliseconds, this has to be
## adjusted using some testing system in the
## real version
## Smaller note values are more probable
        for n in range(motilen):
            motitem = random.randint(0,6)
            if motitem == 0:
                rhythmo.insert(0,48)
            if motitem == 1:
                rhythmo.insert(0,96)
```

FIG. 4b

```
            if motitem == 2:
                rhythmo.insert(0,36)
            if motitem > 2:
                rhythmo.insert(0,24)

## several modifying functions may be called in
## following lines, these functions could be
## switched on or off, if the user would like
## switching mechanism not yet implemented
    rhythmo,motive = motivetrills(rhythmo,motive)
    rhythmo = nonmechanizer(rhythmo)
    print rhythmo
    motive = motiverulessamenes(motive)
    motive = diatonicer(motive)
    motive = motiverules(motive)

## The motive lenght may be changed because of
## transformations, so the "motilen" variable is
## updated:
    motilen = len(motive)
## Main generation loop, processed motive is read
## several times and simultaneously modified
## using transform-system
    for n in range(leng):
        ###TRANSFORMATION FLAG HERE!!!
## if n % 4 == transform[0] % 4:
    if n % motilen == abs(transform[0]):
        motive[n % motilen] = motive[n %
motilen] + transform[1]
    if motive[n % motilen] > transform[0]:
        motive[n % motilen] = motive[n %
motilen] - 12
    if motive[n % motilen] < transform[0]:
        motive[n % motilen] = motive[n %
motilen] + 12
```

FIG. 4c

```
## we test the motive after transformation so as
## to make it more "beautiful"
            motive = motiverules(motive)
            pitch = motive[n % motilen]
            rhyth = rhythmo[n % motilen]

## Following statement is the output-statement,
## by replacing it with ringdata.h type
## outputter, proper data type for output may be
## obtained
           mym.putdata(startp,144,pitch + 79 +
transpo,64,duratio)

## print startp,pitch,duratio
## Following three lines updates the starting
## points and calculates suitable note durations
## non-legato playing) with not too clever way
           startp = startp + (rhyth / 2)
           duratio = rhythmo[(n + 1) % motilen] / 2
           duratio = duratio - randura
           if duratio < 6:
               duratio = 6
           if duratio == 10 :
               duratio = 9

## midi-routines, do not care
    mym.calcdelta()
    mym.putmidi()

## print mym.filename,' - done'

## This function is a heuristic interval
## corrector, which tries to prohibit bad
## sounding intervals like tritonus and sevenths
## This function returns the transformed motive
```

FIG. 4d

```
def motiverules(motive):
    k,intv,na,nb  = 1,0,0,0
    while k < len(motive):
        na = motive[k - 1]
        nb = motive[k]
        if na - nb == -6:
            nb = nb + 1
            motive[k] = nb
        if na - nb == 6:
            nb = nb - 1
            motive[k] = nb

        if na - nb == -10:
            nb = nb + 2
            motive[k] = nb
        if na - nb == 10:
            nb = nb - 2
            motive[k] = nb

        if na - nb == -11:
            nb = nb + 1
            motive[k] = nb
        if na - nb == 11:
            nb = nb - 1
            motive[k] = nb
        k = k + 1
    return motive

def diatonicer(motive):
    scale = []
    k,intv,na,nb  = 1,0,0,0
    tmp,carryflag,protopos = 0,0,0
    majorproto = scaleproto

    while k < len(motive):
        na = motive[k]
        if na % 12 == 3 or na  % 12 == 1 or na %
12 == 6 or na % 12 == 8 or na % 12  == 10:
```

FIG. 4e

```
            na = na - 1
        motive[k] = na
            k = k + 1
    return motive

## This function makes small random inaccuracies
## for the playing to get more "natural" feeling
def nonmechanizer(rhythmo):
    k,intv,na,nb  = 1,0,0,0
    while k < len(rhythmo):
        na = rhythmo[k]
        na = na + random.randint(-3,14)
        rhythmo[k]  = na
        k = k + 1
    return rhythmo

## This rule prohibits successive identical
## intervals, this is necessary, because they
## sound bad in faster tempi
def motiverulessamenes(motive):
    k,intv,na,nb  = 1,0,0,0
    while k < len(motive):
        na = motive[k - 1]
        nb = motive[k]
        if na - nb == 0:
            nb = random.randint(0,24)
            motive[k]  = nb
        k = k + 1
    return motive

## This system changes long note values (i.e. 96)
## to series of alternating pitches
## The series may have its own pitch
## transformation
## the lenght of both motive and rhythmo may be
## changed in a non-foreseeable way
def motivetrills(rhythmo,motive):
    k,kk,intv,na,nb,np  = 0,0,0,0,0,0
```

FIG. 4f

```
    transfo = random.randint(0,4)
    transfob = random.randint(0,4)
    transforva = random.randint(-4,4)
    transforvab = random.randint(-4,4)
    trillen = random.randint(2,16)

    while k < len(rhythmo):
        na = rhythmo[k]
        nb = motive[k]
        np = motive[(k + 1) % len(motive)]
        k = k + 1
        if na == 96:
            rhythmo.remove(96)
            motive.remove(nb)
            for kk in range(trillen):
                rhythmo.insert(k,12)
                if kk % 2 == 0:
                    if transfo == 3:
                        motive.insert(k,nb +
transforva)
                    if transfo < 3:
                        motive.insert(k,nb)
                    if transfo == 4:
                        motive.insert(k,nb -
transforva)

                if kk % 2 == 1:
                    if transfob == 3:
                        motive.insert(k,np +
transforvab)
                    if transfob < 3:
                        motive.insert(k,np)
                    if transfob == 4:
                        motive.insert(k,np -
transforvab)
    return rhythmo,motiv
tst(165,'meca1.mid')
```

FIG. 4g